NOVIA
UNIVERSITY OF APPLIED SCIENCES

Dynamic web application for searching, comparing and editing product data

Armin Vehabovic

Bachelor's Thesis

Information Technology

Vasa 2014

BACHELOR'S THESIS

Author:	Armin Vehabovic
Degree Programme:	Information Technology, Vaasa
Supervisor:	Susanne Österholm

Title: Dynamic web application for searching, comparing and editing product data

---

Date 4.9.2014 Number of pages 28

Abstract

This thesis was commissioned by Wärtsilä Finland Oy, Ship Power. The task was to design a database capable of storing different types of product data. A web application to improve the previously used Excel spreadsheet for comparing competitor engines with Wärtsilä ones, should also be created. Moreover, an administration page was required, where authorized users could administer the web application.

This was mainly done with SQL Server and .NET techniques such as ASP.NET and ADO.NET as well as jQuery for added functionality.

The end product is a dynamic web application where users can search, compare and edit products without being tied to only one product type.

Language: english Key words: ASP.NET, ADO.NET, SQL Server, jQuery

---

EXAMENSARBETE

Författare: Armin Vehabovic

Utbildningsprogram: Informationsteknik, Vasa

Handledare: Susanne Österholm

Titel: Dynamisk webbapplikation för sökning, jämförande och redigering av produktdata

---

Datum 4.9.2014 Sidantal 28

Abstrakt

Detta examensarbete har beställts av Wärtsilä Finland Oy, Ship Power. Uppgiften var att designa en databas som kan lagra data för olika typer av produkter. En webbapplikation skulle också skapas för att förbättra det tidigare använda Excel-dokumentet, som använts för att jämföra konkurrerande motorer med Wärtsiläs. Dessutom var en administratörsida där befogade användare kunde administrera webbapplikationen, nödvändig.

Detta gjordes huvudsakligen med SQL Server och .NET-tekniker såsom ASP.NET och ADO.NET samt jQuery för extra funktionalitet.

Slutprodukten är en dynamisk webbapplikation där användare kan söka, jämföra och redigera produkter utan att vara bundna till en enda produkttyp.

Språk: engelska Nyckelord: ASP.NET, ADO.NET, SQL Server, jQuery

---

OPINNÄYTETYÖ

Tekijä: Armin Vehabovic

Koulutusohjelma: Tietotekniikkaa, Vaasa

Ohjaajat: Susanne Österholm

Nimike: Dynaaminen web-sovellus tuotetietojen etsimistä, vertaamista sekä muokkaamista varten

---

Päivämäärä 4.9.2014 Sivumäärä 28

Tiivistelmä

Tämän opinnäytetyön toimeksiantaja on Wärtsilä Finland Oy, Ship Power. Tehtävänä oli suunnitella tietokanta johon voi tallentaa tietoa erilaisista tuotteista. Oli myös tarvetta web-sovellukselle joka parantaisi aiemmin käytettyä Excel-tiedostoa, millä pystyi vertailemaan kilpailijoiden moottoreita Wärtsilän omiin. Tämän lisäksi oli tarvetta luoda sivu, millä oikeutetut käyttäjät pystyivät ylläpitämään web-sovellusta.

Tämä toteutettiin pääasiallisesti SQL Server ja .NET teknologioiden avulla, kuten ASP.NET ja ADO.NET ja lisätoimintoja varten käytettiin jQuery-teknologiaa.

Lopputuote on dynaaminen web-sovellus, jossa käyttäjät voivat etsiä, vertailla ja muokata tuotteita ilman että he ovat sitoutuneet yhteen tuotetyyppiin.

Kieli: englanti Avainsanat: ASP.NET, ADO.NET, SQL Server, jQuery

---

Table of Contents

1. Introduction

1.1 Employer

Wärtsilä, founded in 1834, started out as a saw mill in the municipality of Tohmajärvi and has come a long way since then. Throughout its 180 years the company has always stayed at the frontier of engineering innovations.

Wärtsilä Finland operates in Helsinki, Vaasa and Turku with the workforce of 3600 professional employees. Globally Wärtsilä operates in more than 200 locations in 70 different countries and with a workforce of approximately 18 700 employees.

The actual employer Ship Power provides ship machinery, propulsion and maneuvering solutions. Ship Power was accounted for approximately 30% of the company's total net sales in 2013. [1]

Figure 1. Wärtsilä headquarters in Helsinki. [2]

1.2 Background

The department needed to compare Wärtsilä engines with the competitors' and therefore an Excel spreadsheet was developed and used. For the Excel spreadsheet to be practical it had multiple sheets with different purposes. The engine data was stored in one of the sheets, which meant that the user could easily edit data about the engines. Two other sheets were built up with macros so users were also able to search and compare the engines.

This project came about when the previously used Excel spreadsheet was deemed not to be the best solution available. It was mainly due to whenever information in the Excel spreadsheet was updated then that Excel spreadsheet had to be sent to all the users. Otherwise those users would compare outdated engine data without even knowing it.

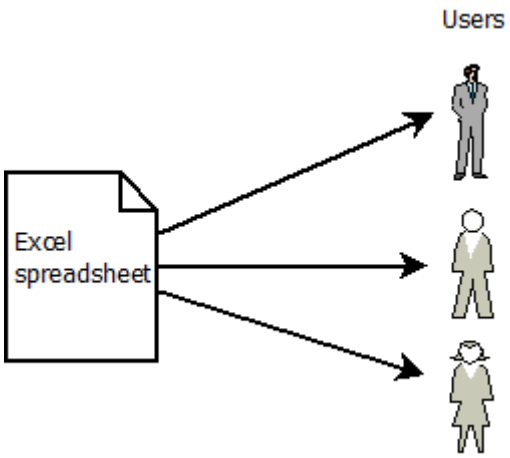


Figure 2. The distribution of the Excel spreadsheet.

1.3 Task

The task was to create a dynamic web application where users can search for products and be able to compare products within the same product type. The comparison was to be designed so the users could see the differences in a fast and easy manner.

In order to do this it was important to design a database that can store product data without being tied to just one product type. It was also important to keep information redundancy minimized for the data retrieval to be responsive.

An administration page also had to be created where authorized users could mainly edit product data but also change the search filters and how the comparison was displayed.

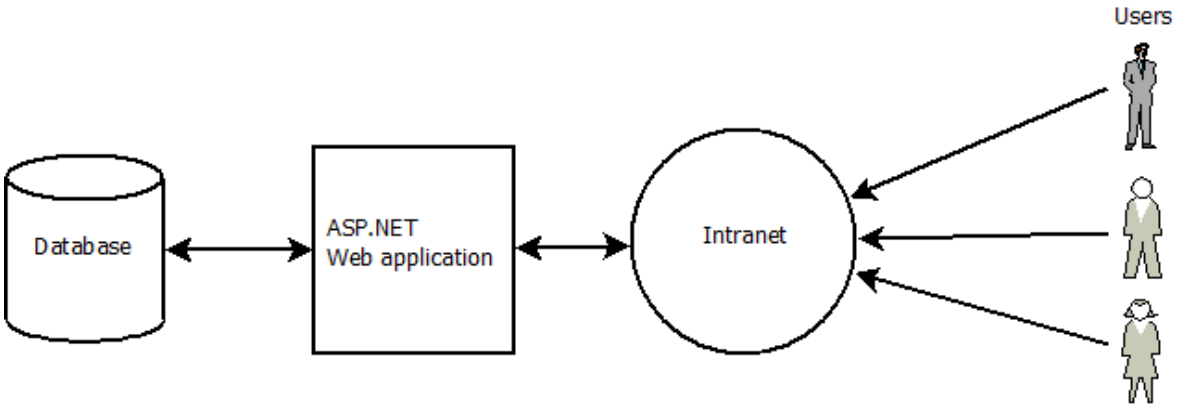


Figure 3. The distribution of the web application.

The main idea of the web application was that whenever a user visited the web application he would always search and compare with product data that is up-to-date.

1.4 Requirements

The requirement was that the web application would be developed to be as dynamic as possible. This meant that the web application could take any product type and have the capability to search, compare and edit product data.

This was mainly done to make the web application set for the future which the Excel spreadsheet was not capable of. The Excel spreadsheet had been built with only the product type engine in mind.

Requirements that needed to be met:

- Use of .NET techniques
- Web application optimized for Internet Explorer 9
- Dynamic search functionality
- Ability to compare products within the same product type
- Ability to export the comparison to Excel
- Administration tool for authorized users to update the information

2. Theory

This chapter serves as a short introduction to the programming languages, tools and frameworks used in this thesis.

2.1 SQL Server

SQL Server developed by Microsoft is a relational database management system. Its main function is to store and retrieve data requested by other applications, be it over the global internet or within the internal network. [3]

For testing purposes a local instance of SQL Server Express was used. It's free to download but there are some technical restrictions which make it only suitable for smaller scale deployments. [4]

2.1.1 T-SQL (Query Language)

T-SQL short for *Transact - Structured Query Language* is basically an extension of the query language found in SQL Server. It's capable of delivering more features to the language than would otherwise be possible. [5]

```
CREATE TABLE Test
(
    Id int IDENTITY(1,1) PRIMARY KEY,
    Name nvarchar(50),
    CreatedDate datetime DEFAULT GETDATE()
);

INSERT INTO Test(Name) VALUES ('John Doe');
INSERT INTO Test(Name) VALUES ('Tauno Tavallinen');

select * from Test;
delete from Test where DATEPART(D,CreatedDate) = (DATEPART(D,GETDATE())-1);
```

Code example 1. T-SQL example

The T-SQL code example 1 creates a table named “Test”. When inserting data the first row gets Id 1, Name “John Doe” and CreatedDate today’s date. The second row gets Id 2, Name “Tauno Tavallinen” and CreatedDate today’s date. The select command displays all the rows in the table. The last command deletes all the rows which are 1 day old. The DATEPART function extracts the date part of the date value given as parameter.

2.1.2 SQL Server Agent

SQL Server Agent, one of the services available in SQL Server, allows automating some of the administration tasks and making tasks run by a user defined time schedule. In addition to automation, SQL Server Agent is able to monitor changes and send out alerts depending on what it’s set to do. [6]

To be able to use SQL Server Agent the full version of SQL Server was needed. SQL Server Agent is not included in SQL Server Express due to earlier mentioned restrictions in chapter 2.1.

2.2 .NET Framework

.NET Framework primarily developed for Windows by Microsoft is a software framework which includes a large class library known as Framework Class Library (FCL). FCL provides language interoperability between several programming languages. This means that each programming language can use code written in several other programming languages.

Common Language Runtime (CLR), a virtual machine component of .NET Framework, is responsible for managing the execution of applications written for .NET. However, for CLR to be able to execute the code it first needs to be translated into a code that CLR understands. Rather than compiling the code into CPU- or platform-specific object code, it is handled by the CPU- and platform-independent Common Intermediate Language (CIL). After CIL has converted the code into so-called bytecode, which can be run by CLR, the CLR converts the compiled code into machine instructions that the PC can execute.

The CIL and CLR are a part of the open specification Common Language Infrastructure (CLI) which describes the process of the executable code and the runtime environment. [7]

2.2.1 ADO.NET

ADO.NET provides a direct method for accessing persistent data within the .NET Framework. It is used for connecting to data sources and retrieving, handling or updating the data and it is able to do it in a connected or a disconnected state.

Accessing the data from a database in a connected state means that the data is retrieved while there is an open connection. Meanwhile for a disconnected state, the data is retrieved and stored in dataset. The data in a dataset can be manipulated and displayed even after the connection has been closed. [8]

```
private DataSet ExampleFunction()
{
    DataSet dataSet = new DataSet();
    using (SqlConnection con = new SqlConnection(GetDbInfo()))
    {
        using (SqlCommand cmd = con.CreateCommand())
        {
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "SELECT * FROM Products WHERE ProductType='Engine'";
            using (SqlDataAdapter da = new SqlDataAdapter(cmd))
            {
                using (DataTable dataTable = new DataTable())
                {
                    da.Fill(dataTable);
                    dataSet.Tables.Add(dataTable);
                }
            }
        }
    }
    return dataSet;
}
```

Code example 2. Using ADO.NET to get all the engine products and return them in a dataset.

Another option would have been to use Entity Framework instead of ADO.NET. Entity Framework enables the programmer to work with domain-specific objects and properties, thus eliminating most of the data-access code needed when using ADO.NET.

The way Entity Framework works is by having the user create an Entity Data Model. It can be generated from an existing database which is called Database-first. A second way would be to generate the database from code which is called code-first. The Entity Data Model is responsible for handling the underlying database tables and relationships. ADO.NET was chosen over Entity Framework due to programming experience. [9]

2.2.2 ASP.NET

ASP.NET developed by Microsoft is a part of .NET Framework and gives the user suitable tools to build dynamic web applications. This is done by combining .NET languages with the more traditional HTML and CSS with minimal coding.

The way it works is when the page is requested by a browser, the ASP.NET renders the HTML to the requesting browser. Depending on the browser it renders the markup accordingly. Browsers may not have the same support for the tags used and it may differ from browser to browser.

There are mainly two alternatives when making web pages with ASP.NET. These are ASP.NET Web Forms and ASP.NET MVC.

With ASP.NET Web Forms you are able to separate the HTML code and other user interface code from the application logic. It makes ASP.NET Web Forms easy to use and for beginners it's much easier to get a grasp of.

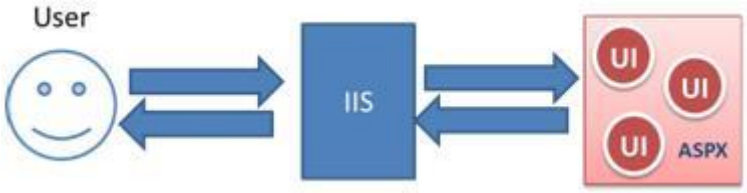


Figure 4. A simple explanation of how ASP.NET Web Forms works. [10]

ASP.NET MVC (Model-View-Controller) is an open-source framework which just describes the way of splitting up the code. As the name of the abbreviation suggests, the code is split into Model, View and Controller. Model is the business layer, View the display layer and Controller is the input control. The benefit of doing it this way is that it becomes easier to replace one of the parts. [11]

Out of the two ways of developing a web application in ASP.NET, the choice fell on ASP.NET Web Forms. This was mainly because only an older version of MVC was available.

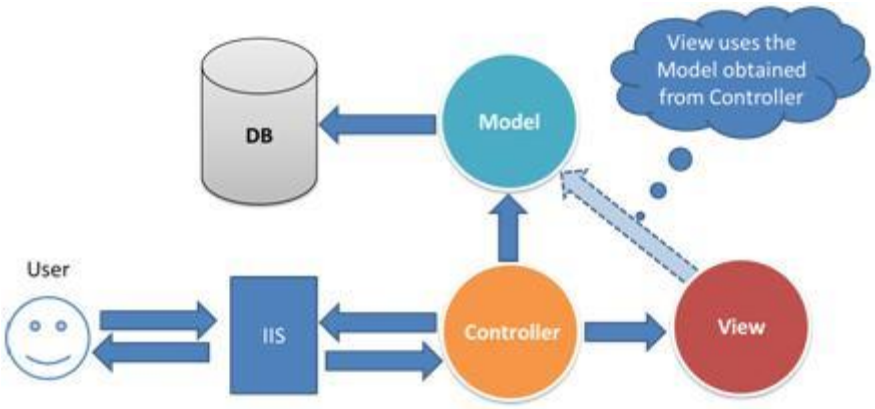


Figure 5. A simple explanation of how ASP.NET MVC works. [10]

2.3 C#

C# is an object-oriented programming language developed by Microsoft initially for the .NET Framework but it was later approved as its own standard. C# is intended to be simple, general-purpose and modern. [12]

```
class TestCSharp
{
    static void Main()
    {
        Console.WriteLine("Hello World");
    }
}
```

Code example 3. A short C# example where Hello World is written to the console.

ASP.NET mainly supports applications written in C# and VB.NET which are .NET compatible languages. C# was chosen due to programming experience but VB.NET could just as well have been chosen.

VB.NET is quite similar to C# but there are a couple of main differences. Instead of curly brackets, group statements are closed by keywords. Statements are terminated by a new row instead of a semicolon, which is used by several programming languages. [13]

```
Module TestVb

    Sub Main()
        Console.WriteLine("Hello World")
    End Sub

End Module
```

Code example 4. A short VB.NET example similar to the C# example.

2.4 HTML

HTML, short for *HyperText Markup Language*, is the primary used markup language when creating websites. Its elements are the building blocks of all websites.

HTML is considered a markup language and not a programming language because it consists of tags which usually come in pairs, opening and closing tags. A web browser reads the HTML document and depending on the web browser, the site may be rendered in a different way. Each web browser may support the markup language in a different way. [14]

```
<html>
    <head>
        <title>HTML Example</title>
    </head>
    <body>
        <p>Small example of how HTML works.</p>
    </body>
</html>
```

Code example 5. A small code example of how HTML works.

In addition to the regular HTML tags, ASP.NET offers a set of its own unique components. These components look similar but some of the ASP.NET components offer richer features with less coding involved than would be otherwise possible with regular HTML elements. [15]

```
<panel>
    <h2>Example</h2>
    <p>Combining HTML tags with ASP.Net HTML tags.</p>
    <asp:CheckBox ID="checkbox1" runat="server"/>
    <asp:Label ID="label1" runat="server" Text="Checkbox"></asp:Label>
    <br />
    <asp:Button ID="button1" runat="server" Text="Save"/>
</panel>
```

Code example 6. A small code example combining regular HTML tags with ASP.NET tags.

2.5 CSS

CSS, short for *Cascading Style Sheets*, is primarily designed for separating the presentation of a document from the document content. This means that multiple pages can share the formatting, thus reducing the complexity and repetition of the structural content. Therefore, improving the content accessibility and providing more flexibility and control in the overall presentation characteristics. [16]

```
p {
    color:white;
    background-color:black;
    width:300px;
}
```

Code example 7. A CSS example which manipulates the paragraph element found in code example 5.

2.6 jQuery

A widely used cross-platform JavaScript library called jQuery is designed to simplify the client-side scripting of HTML. With jQuery you are able to create animations, handle events and much more, making it easy to create powerful dynamic webpages.

```
$("#textbox1").text("A jQuery example");

$("#textbox2").hide();
```

Code example 8. Using jQuery to change the text of textbox1 and hide the textbox2.

The main advantage of jQuery over JavaScript is that you can do more with less coding involved. [17]

3. Solution

3.1 Designing the product table

The product table needed the capability to hold product data with different product types while keeping the information redundancy minimized. This meant that it should be possible to use the same columns in the product table for completely different products. So designing the table which was needed to store the products for this web application was trickier than in normal cases.

After trying out multiple database designs the chosen design was a pretty simple one. The product data is stored in a table where the column names are just named after the data type the columns represented. This was just due to there being several columns of the same data type so after each column name there was an incrementing number to keep the names unique.

To be able to use the products table there needed to be another table where the corresponding columns had a name and a unit. By doing it that way the product types table was kept small and each product type used only one row. This meant that all products attached to the same product type had the same data stored per column.

A product type in the product types table could have many products connected to it but a product could only be connected to one product type.

P_id	ProductType	Int	Float	Nvarchar	Nvarchar2
1	Engine	4	1,3	Wärtsilä	20
2	Engine	6	1,2	Wärtsilä	26
3	Book	30	NULL	Thesis	Amin
4	Book	500	NULL	Diagrams	Amin

ProductType	Int	Int_unit	Float	Float_unit	Nvarchar	Nvarchar_unit	Nvarchar2	Nvarchar2_unit
Engine	N# Cyl	NULL	Coeff.	times	Engine Designer	NULL	Engine Series	NULL
Book	Pages	NULL	NULL	NULL	Type	NULL	Author	NULL

Figure 6. An insight on the products table and the product types table.

As can be seen in figure 6 just looking at the two tables separately doesn't make much sense. By writing a query and joining the two tables it's much easier to see how it works.

	ProductType	Int	Int	Int_unit
1	Engine	N# Cyl	4	NULL
2	Engine	N# Cyl	6	NULL
3	Book	Pages	30	NULL
4	Book	Pages	500	NULL

	ProductType	Float	Float	Float_unit
1	Engine	Coeff.	1,3	times
2	Engine	Coeff.	1,2	times
3	Book	NULL	NULL	NULL
4	Book	NULL	NULL	NULL

Figure 7. The columns Int and Float from the result of joining the two tables products and product types.

	ProductType	Nvarchar	Nvarchar	Nvarchar_unit
1	Engine	Engine Designer	Wärtsilä	NULL
2	Engine	Engine Designer	Wärtsilä	NULL
3	Book	Type	Thesis	NULL
4	Book	Type	Diagrams	NULL

	ProductType	Nvarchar2	Nvarchar2	Nvarchar2_unit
1	Engine	Engine Series	20	NULL
2	Engine	Engine Series	26	NULL
3	Book	Author	Armin	NULL
4	Book	Author	Armin	NULL

Figure 8. The columns Nvarchar and Nvarchar2 from the result of joining the two tables products and product types.

As can be seen in figure 7 and 8 it's much easier to understand the concept behind that two tables. The sole purpose of having the products table split into two tables is that it can support products with different product types.

3.1.1 Automating product data retrieval

Having a separate product database for the web application meant that the product data needed to be updated by hand. To automate some of the product data gathering it was decided that the possibility of using an internal Wärtsilä database should be looked into.

This was done by using SQL Server Agent to make a job which automated the process. The job's first step was to remove the old Wärtsilä engines from the product database of the web application.

The second step was to select the necessary data of the Wärtsilä engines from the internal Wärtsilä database and insert them into the product database of the web application. The job was scheduled to run once a week and in case of any error an email would be sent out to an administrator who could look into it.

3.2 Homepage

The homepage of the web application informed the user that the website was optimized for Internet Explorer 9 but would work as well for other browsers. The differences between the browsers were only minor because the workarounds chosen worked in all the browsers. More about one of these workarounds can be read in chapter 3.3.

As well as informing the user about browser compatibility the homepage also showed the last changes made to products. The last 7 changes made to any products were displayed and this was just so the user could see if any changes have been made lately to any of the products.

3.3 Searching products

The inspiration for the user interface came from the previously used Excel spreadsheet. As the Excel spreadsheet was only used for the product type engines, there had to be some changes to make it work with multiple product types.

To make it easy and functional for multiple product types, a simple dropdown menu was chosen where the user could choose which product type he wanted to search from. When the item in the dropdown menu changed, an event was fired that executed code behind. This loaded up the new search filters for the selected product type.

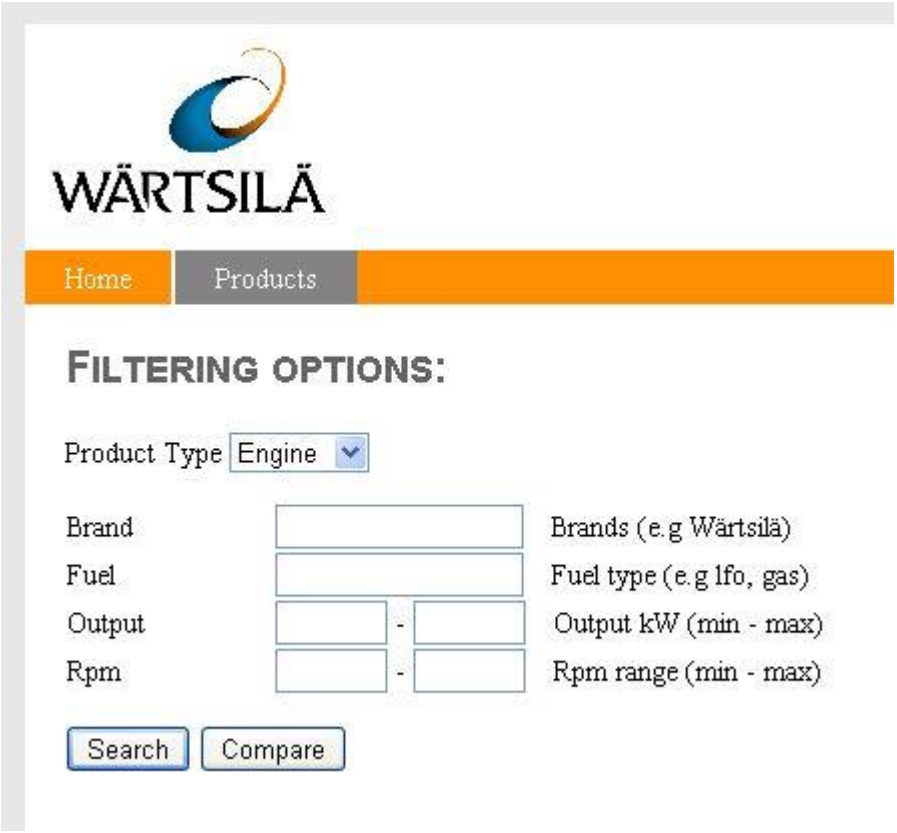


Figure 9. Filtering options for the product type Engine

To further improve the user experience an autocomplete function was developed which would display a dropdown menu as the user typed and he could then choose the item he wanted. This didn't only work for one item. As items were chosen the autocomplete function automatically added a delimiter after the first item and the user could continue typing and add more items. This was done by creating an ASP.NET Webservice which handled getting the information from the database and a jQuery script found on the net modified to suit the needs. [18, 19]

Figure 10. The implemented jQuery autocomplete feature.

After selecting a search criterion the products were displayed underneath the search filters. For each product there was a checkbox that could be ticked if the user wanted that product to be added for comparison.

After having a demo for a few people, it was pointed out that it would be much easier if the header of the search table was in a fixed position while the rest of the table would remain scrollable. This would make choosing the products one wanted to compare easier and it would improve the overall user experience.

Getting the header in a fixed position proved to be a bit more difficult than normally because of the choice of primarily developing the web application for Internet Explorer 9. The reason for this is that there is no easy fix like there is in the other browsers for a table with a fixed header. This is due to various bugs in Internet Explorer 9 but if there is something to learn that is that there is always a workaround. The workaround chosen for this problem was to use a jQuery script from the net that was heavily modified to suit the needs. The purpose of the script was to split the table into two, therefore, making the first table act as a fixed header while maintaining a scrollable view of the second table with the products. [20]

3.4 Comparison

After selecting products the user could advance to the comparison by clicking the designated button. It redirected him to the comparison page if there were any products chosen, otherwise it wouldn’t redirect anywhere. As with the page for searching products, the inspiration for the comparison came from the previously used Excel spreadsheet.

The comparison table was aligned vertically, which meant that the first column of each row informed what the row was about. Additional columns were the products that the user chose for comparison. [21]

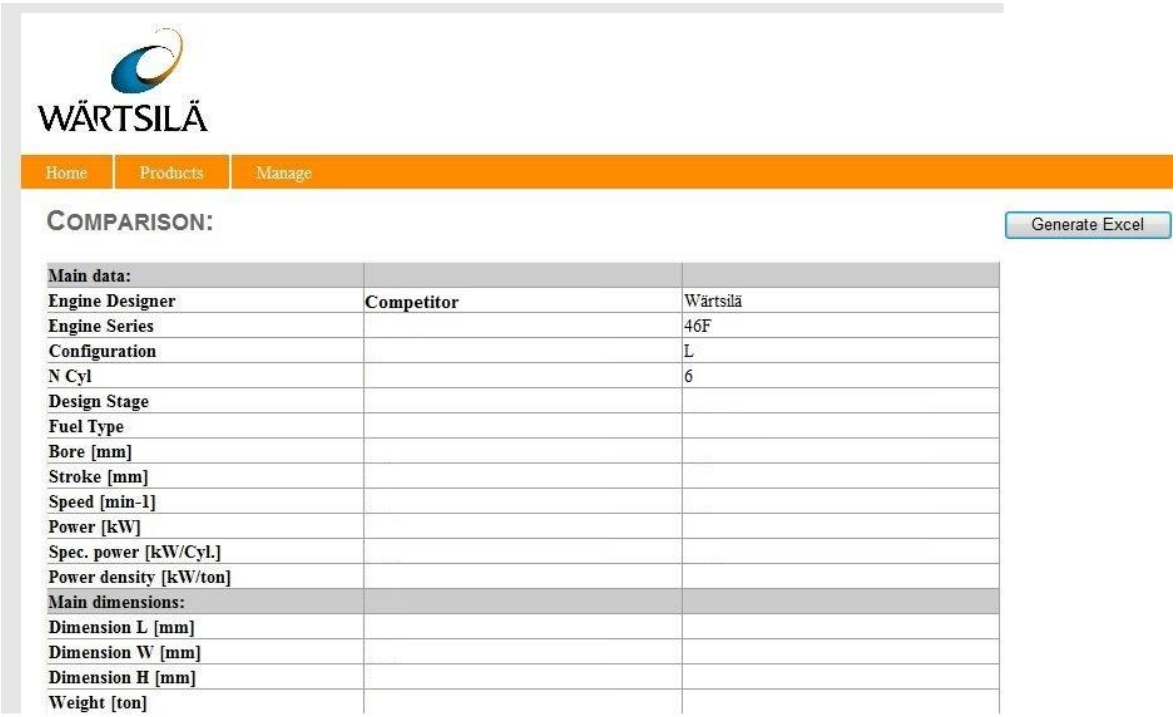
As the data in the database was not organized in the order the user wanted it displayed, there had to be a way to make it always show in a user defined order. As mentioned earlier in chapter 3.1, each product type had columns which said what the corresponding columns in the products table were. So as well as saying what those columns were, it also said in what position the columns should come. When rendering the comparison site it first loaded up the columns in the order defined by the position columns, then displayed them. This made the columns always show in the same order.

To further improve the user experience it was suggested that it would be easier to navigate the comparison page if the items were divided into groups. This would mean that the user could easily just move to the group he was interested in.

This was easily done by using the same concept as for the positioning of the columns. An additional field named “grp” which said what group each column belonged to, was added. If the group field wasn’t empty and the following group fields were, then those columns belonged to the same group.

The next group field that wasn’t empty was the start of the next group and so on. When the site rendered it changed the CSS class of the group fields that weren’t empty. This made it easy to manipulate the group headers so the user could easily spot the categorization of the groups.

By doing it this way there were some drawbacks. If there was nothing defined for one or more columns, there would be many empty items. This would clutter down the comparison unnecessarily. Only if all the columns of a row were empty it wouldn’t show the row. Furthermore, only if all the rows belonging to the same group were empty, then they would be hidden.



Main data:		
Engine Designer	Competitor	Wärtsilä
Engine Series		46F
Configuration		L
N Cyl		6
Design Stage		
Fuel Type		
Bore [mm]		
Stroke [mm]		
Speed [min-1]		
Power [kW]		
Spec. power [kW/Cyl.]		
Power density [kW/ton]		
Main dimensions:		
Dimension L [mm]		
Dimension W [mm]		
Dimension H [mm]		
Weight [ton]		

Figure 11. A censored comparison page.

Please note that figure 11 has been censored so it doesn't represent the real comparison page. The button "Generate Excel" in figure 11 exports the comparison table to Excel.

These changes made to the comparison page meant that a user could easily and in a fast manner find the information he was looking for. This meant that the overall user experience was improved.

3.5 Security

To be able to authenticate users and grant access to the administration page only to authorized users required a login solution. There are several alternative login solutions in ASP.NET. To keep it simple a login solution where the user had a login name and a password couldn't be used. It wouldn't be positive in any way to have another application with login name and password. The only solution was to implement an intranet login which would automatically recognize the user and grant access to the administration page.

To be able to tell which users should be granted access to the administration page a table was created in the database where the usernames were stored. Whenever a user visited the website, the intranet login solution checked the user's username. If the username was found in the users table, the user was granted access to the administration page. Otherwise he would just have access to the search and compare. [22]

3.6 Administration page

The site being dynamic as it was meant that there were many things that could be modified by an authorized user. The administration page had to be designed so it would be easy enough to use but at the same time it should be flexible enough to have multiple options for modifying parts of the site.

Because of how the Excel spreadsheet was built, it didn't have an administration interface. Because of this there was nothing to go on, when designing the administration page. The final design chosen for the administration page had quite a dynamic design.

All the different options were per product type, which meant that any products added or any changes made were to the product type that was selected. To keep it simple the dropdown menu from the products page was reused so the authorized user could easily recognize it.

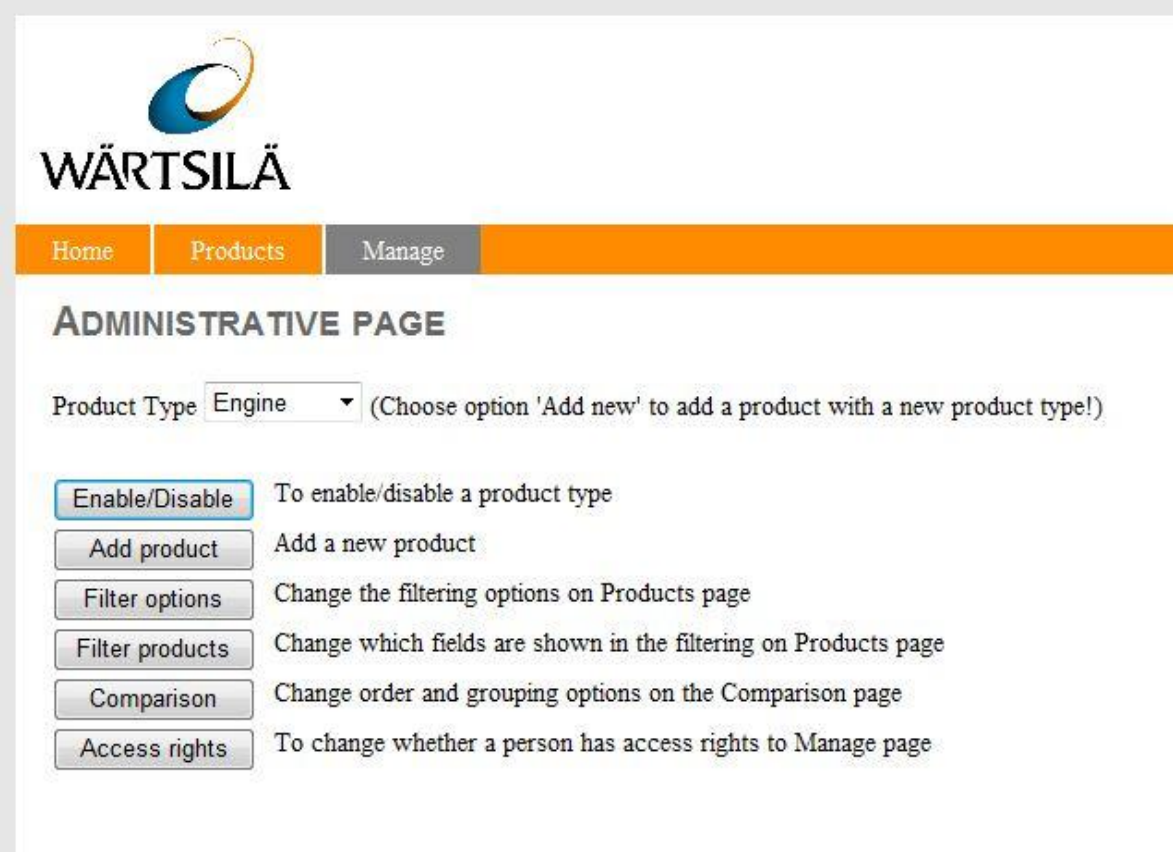


Figure 12. Overview of the administration page.

For simplicity's sake the explanation of the different options will be in the order shown in figure 12.

To add a new product type the user had to choose the option "Add new" in the dropdown menu. It would display a textbox where the user could enter the name of the product type and save it.

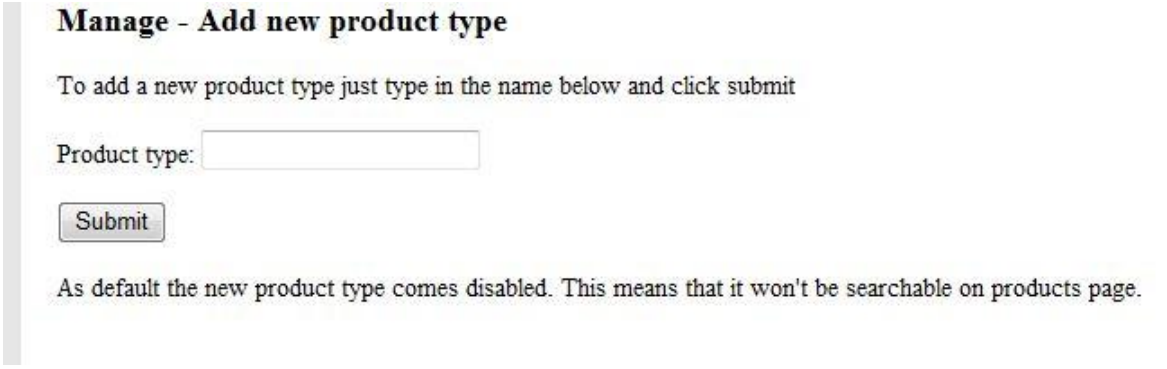


Figure 13. Adding a new product type.

Using the option Enable/Disable simply enabled or disabled a product type. A disabled product type was not visible to the regular users visiting the site but would remain on the administration page for future use. To not accidentally remove a product type, the option to delete a product type was hidden from plain view. Only when the user disabled a product type, would the option to delete be displayed.

As default when creating a new product type the product type was disabled. This was because the product type was empty. It wasn't possible to enable the product type until a couple of things had been completed. The product type needed to have a product added, as well as search filters, and which columns should be shown in the search table were needed to be able to enable the product type.

Figure 14. A disabled product type.

As can be noted in figure 14, removing a product type will remove every product and all the settings for it as well, and this was because of how the database was built.

There was a small difference between when adding a product to a product type which already had products and to a product type which had no products. It was mainly because none of the columns were taken for a product type which had no products. This meant that it was up to the authorized user to set the standards for the new product type.

Inserting a product to a new product type meant that the data of all the columns were checked to see if it was a number, decimal or plain text. Afterwards, the data was inserted in the first best column that matched the data type. Furthermore the columns' position was set in the same order as they were entered.

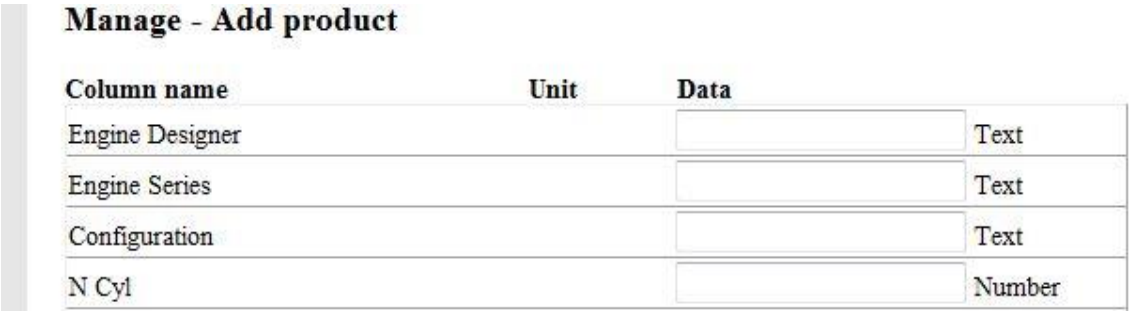


Figure 15. First few rows of the adding a product option.

As can be seen in figure 15, adding a product to a product type which already had products meant that the column names were already defined and the data type of the fields was also defined. For a better user experience the data types were written after the data textboxes. Because of this there were no losses of information. If a user tried to enter a decimal into a number data field, it would write out an error message instead of accepting it.

Once a product had been added to a new product type it didn't mean that no more columns could be added in the future for the product type. It was possible to do this because all the columns were shown and not just those that were being used. To add a new column to a product type, the authorized user only had to scroll down to an unused row. If there were no empty rows, it meant that all the columns were already taken.

There were just a certain number of different data type columns available so the user couldn't use all the available columns for one data type. When the user tried to save the product the web application checked whether the limit had been exceeded on any of the data types. If the limit had been exceeded, the web application wrote an error message.

The option to edit or remove a product was moved from the administration page to the products page. This was because it would make a negative impact on the user experience if it had been left on the administration page.

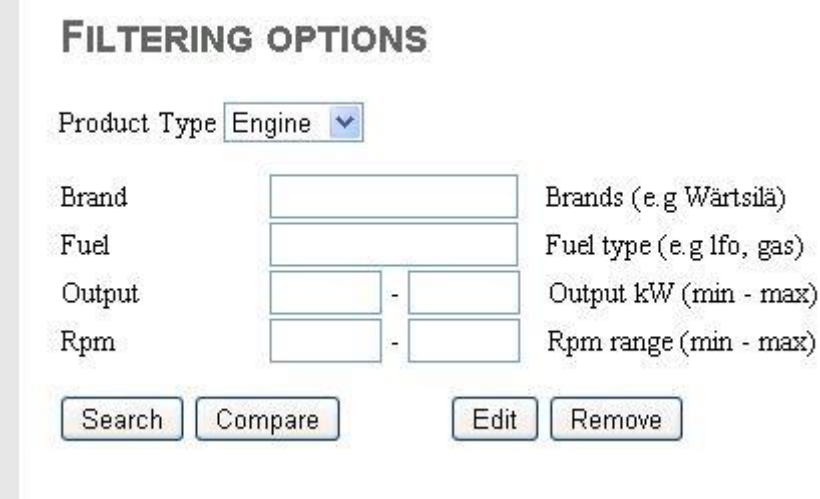


Figure 16. Edit and remove option on products page.

The edit and remove option could only be seen by authorized users. This meant that everybody could use the filters to search for products. Only authorized users could choose to edit or remove the filtered products.

The “Filter options” option was for changing the search filters on the products page. It was a simple and straightforward option.

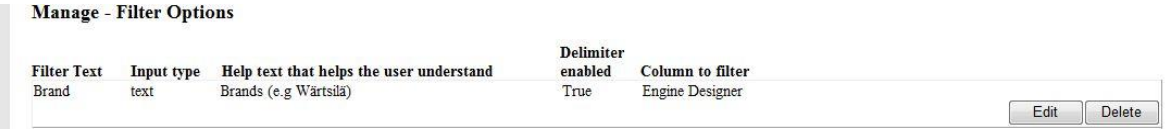


Figure 17. Picture of the first search filter for the product type engine.

As can be seen in figure 17 a user was able to change all the small things. The input type was just for changing whether the search filter would be a simple textbox or a more advanced min – max field.

The delimiter could only be enabled for a simple textbox. With the delimiter set to true the autocomplete feature on the search page was enabled. The delimiter wouldn't benefit the user experience in any way if it could be used in a min – max field. [23]

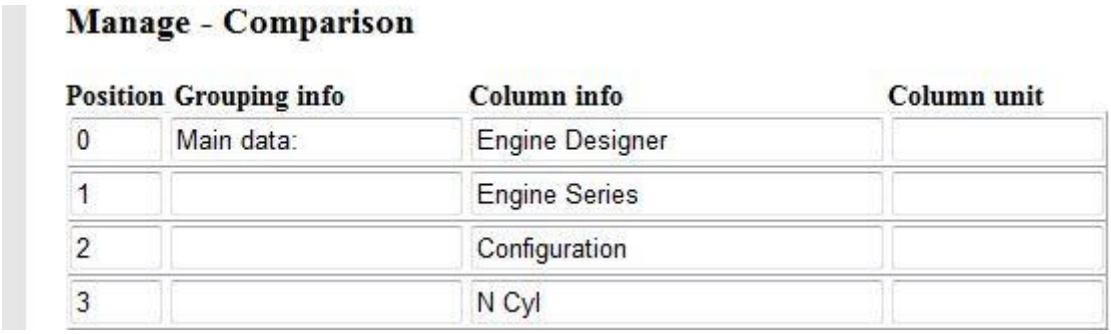
Choosing which column the search filter should apply to was easily achieved by placing the names of the used columns in a dropdown menu. The dropdown menu could only be seen when the search filter was in edit mode. By doing it this way, it was possible to make the search filters very flexible.

There couldn't be more than 4 filters for a product type because this was considered enough. This limitation was set because it wouldn't clutter down the products page unnecessarily. If there were fewer than 4 search filters for a product type, a button to add another search filter was visible.

The "Filter Products" option was for changing which fields should be displayed in the search table. There was just a simple checkbox for each column which could be checked if the column should be shown. The columns in the search table were in the order declared by the position columns in the product type table.

Figure 18. Small peek at the filter products option.

As explained in chapter 3.4, the columns came in a user defined order and the columns were categorized into groups. With the comparison option it was possible to change the order of the columns and how the groups were categorized.



Manage - Comparison

Position	Grouping info	Column info	Column unit
0	Main data:	Engine Designer	
1		Engine Series	
2		Configuration	
3		N Cyl	

Figure 19. Showing the comparison option.

The option “Comparison” was simple enough with only one check implemented. The user was not able to place multiple columns in the same position.

Finally with the “Access rights” option you could select which users should have access to the administration page. To be able to see the option “Access rights” the user had to be a super user. A super user was a trusted user who was able to set the access rights for users, as well as add other users as super users.

4. Result and discussion

4.1 Results

The result is a web application where users within the internal Wärtsilä network can dynamically search for products, select and compare the wanted products and export the comparison table to Excel.

The users are authenticated by the intranet login solution which gives them access to the administration page. On the administration page the authorized users can add and edit products, as well as administer the web application.

4.2 Discussion

The web application turned out better than I first expected mainly because there was plenty of time to develop the web application. I do believe that the users will be pleased with the web application.

The web application was published within the internal network and users started using it a couple of weeks after I was done with the thesis. Because of this I can’t say how the response was when it was first published.

During the development of the web application it was regularly tested by a few users, who usually had only bugs to report and occasionally some suggestions which would improve the user experience. Any suggestion of improvement was taken into consideration and if deemed appropriate later implemented.

The part that took the longest to develop was the administration page. It was mainly because of how the web application was built. This meant that much of it could be changed. Authorized users needed an easy way to administer it all.

4.3 Future development

One of the things that I think can be improved is the “fixed header” solution that I described in chapter 3.3. The jQuery script was fast if the products searched for weren’t too many. The main concern was that if the search returned too many products, it would make the site load more slowly and it could take a couple of seconds to finish loading. It really depends on the average number of products returned for each search. This is something that has to be tested by many users to see their response.

As it is right now an authorized user can only edit one product at a time and this could perhaps be improved so it’s possible to edit more than one product at a time.

The option “Filter products” could be improved, so that an authorized user is able to set by which column the data should be ordered in the search table, because with different product types it’s impossible to set only one column which the data should be ordered by.

References

[1] “Wärtsilä” [Online]. Available: http://www.wartsila.com [Accessed 4 August 2014].

[2] “Wärtsilä” [Online]. Available: http://en.wikipedia.org/wiki/W%C3%A4rtsil%C3%A4 [Accessed 4 August 2014].

[3] “Microsoft SQL Server” [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_SQL_Server [Accessed 7 July 2014].

[4] “SQL Server Express” [Online]. Available: http://en.wikipedia.org/wiki/SQL_Server_Express [Accessed 7 July 2014].

[5] “Transact-SQL” [Online]. Available: http://en.wikipedia.org/wiki/T-SQL [Accessed 7 July 2014].

[6] “SQL Server Agent” [Online]. Available: http://technet.microsoft.com/en-us/library/ms189089%28v=sql.105%29.aspx [Accessed 9 July 2014].

[7] “.NET FRAMEWORK” [Online]. Available: http://en.wikipedia.org/wiki/.NET_Framework [Accessed 9 July 2014].

[8] “ADO.NET Overview” [Online]. Available: http://msdn.microsoft.com/en-us/library/h43ks021%28v=vs.110%29.aspx [Accessed 10 July 2014].

[9] “Entity Framework” [Online]. Available: http://en.wikipedia.org/wiki/Entity_Framework [Accessed 12 August 2014].

[10] “WebForms vs. MVC” [Online]. Available: http://www.codeproject.com/Articles/528117/WebForms-vs-MVC [Accessed 12 August 2014].

[11] “ASP.NET Overview” [Online]. Available: http://msdn.microsoft.com/en-us/library/4w3ex9c2%28v=vs.90%29.aspx [Accessed 15 July 2014].

[12] “C Sharp (programming language)” [Online]. Available: http://en.wikipedia.org/wiki/C_Sharp_(programming_language) [Accessed 15 July 2014].

[13] “Visual Basic .NET” [Online]. Available: http://en.wikipedia.org/wiki/Visual_Basic_.NET [Accessed 11 August 2014].

[14] “HTML” [Online]. Available: http://en.wikipedia.org/wiki/HTML [Accessed 14 July 2014].

[15] “ASP.NET Web Forms | The ASP.NET Site” [Online]. Available: http://www.asp.net/web-forms [Accessed 14 July 2014].

[16] “Cascading Style Sheets” [Online]. Available: http://en.wikipedia.org/wiki/Cascading_Style_Sheets [Accessed 14 July 2014].

[17] "jQuery" [Online]. Available: http://en.wikipedia.org/wiki/JQuery [Accessed 16 July 2014].

[18] "Autocomplete" [Online]. Available: http://jqueryui.com/autocomplete/#multiple [Accessed 1 July 2014].

[19] "Populate jQuery AutoComplete TextBox from Database using Web Service in ASP.NET" [Online]. Available: http://aspsnippets.com/Articles/Populate-jQuery-AutoComplete-TextBox-from-Database-using-Web-Service-in-ASPNet.aspx [Accessed 3 July 2014].

[20] "Add vertical scrollbar to GridView in ASP.NET" [Online]. Available: http://www.aspsnippets.com/Articles/Add-vertical-scrollbar-to-GridView-in-ASPNet.aspx [Accessed 25 June 2014].

[21] "Displaying vertical rows in a GridView" [Online]. Available: http://aspdotnetcodebook.blogspot.fi/2008/04/displaying-vertical-rows-in-gridview.html [Accessed 16 June 2014].

[22] "ASP.NET authentication and authorization" [Online]. Available: http://www.codeproject.com/Articles/98950/ASP-NET-authentication-and-authorization [Accessed 11 June 2014].

[23] “Delimiter” [Online]. Available: http://en.wikipedia.org/wiki/Delimiter [Accessed 21 July 2014].